华中科技大学

计算机视觉课程实验报告

实验一：基于前馈神经网络的分类任务设计

姓　　名：　　李嘉鹏

学　　院：　　计算机科学与技术学院

专　　业：　　数据科学与大数据技术

班　　级：　　大数据 2101 班

学　　号：　　U202115652

指导教师：　　刘康

2023 年 12 月 16 日

目　录

实验一 基于前馈神经网络的分类任务设计

1.1 Introduction（实验简介）

1.1.1 Background（实验背景）

前馈神经网络（又称为全连接神经网络、多层感知器）是计算机视觉领域中常见的一种网络结构。其特征在于：各神经元分别属于不同层，层内无连接；相邻两层之间的神经元全部两两连接；信号从输入层向输出层单向传播。前馈神经网络的基本结构如图 1 所示。

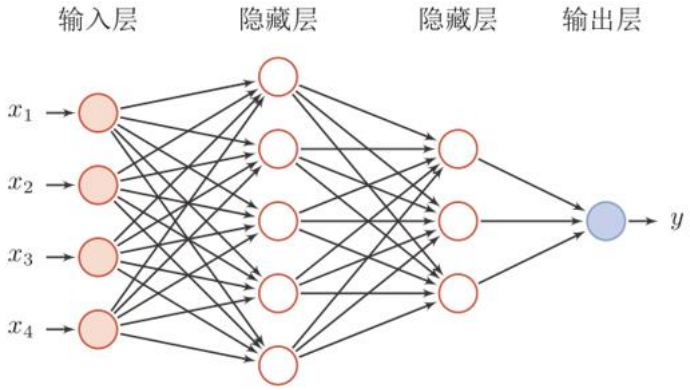

图 1：前馈神经网络结构

前馈神经网络信息传播的公式为：$z^{(l)} = W^{(l)}a^{(l-1)} + b^{(l)}$，$a^{(l)} = f_l(z^{(l)})$，其中 $f_l(\cdot)$ 为第 l 层神经元的激活函数，$W^{(l)}$ 为第 $l-1$ 层到第 l 层的权重矩阵，$b^{(l)}$ 为第 $l-1$ 层到第 l 层的偏置，$z^{(l)}$ 为第 l 层神经元的净输入，$a^{(l)}$ 为第 l 层神经元的输出。前馈神经网络可以在多种分类任务上取得较好的性能，例如一元线性回归、手写体数字识别等。

1.1.2 Motivation（实验目的）

本实验是基于前馈神经网络的分类任务，要求设计一个前馈神经网络，对数据完成分类。

已知 dataset.csv 数据集包含四类二维高斯数据和它们的标签。要求设计至少含有一层隐藏层的前馈神经网络，预测二维高斯样本 $(data_1, data_2)$ 所属的分类

label 。对于该数据集，需要首先进行随机排序，然后选取 90%用于训练，剩下10%用于测试。可以任选深度学习框架，尝试不同的网络层数、不同的神经元个数、使用不同的激活函数等，观察网络性能。

1.2 Proposed Method（实验设计与步骤）

1.2.1 Network Architecture（神经网络架构）

我采用的深度学习框架为 Tensorflow。

实验中的神经网络选用了一个包含 2 个隐藏层的前馈神经网络。第一个隐藏层具有 32 个神经元，第二个隐藏层具有 16 个神经元，这两个隐藏层使用的激活函数可以是 softmax、relu 或 sigmoid 等。输出层具有 4 个神经元，使用 softmax 激活函数进行多分类，类别分别为 0、1、2、3，代表四种不同类型的数据。

神经网络的定义代码如下所示。注意，这里仅代表一种情形下的网络架构，实际上网络层数、激活函数、各层的神经元个数均可修改，不同参数下对应的结果将会在 1.3 节中给出。

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(32, activation='softmax', input_shape=(2,)),
    tf.keras.layers.Dense(16, activation='softmax'),
    tf.keras.layers.Dense(4, activation='softmax')
])
# 网络层数、激活函数、神经元个数可修改
```

由于分类任务中最常用的损失函数为交叉熵损失函数，因此本实验中我使用了 sparse_categorical_crossentropy 作为损失函数来适应多分类问题。同时，我还使用了 adam 作为优化器。训练模型时每个 epoch 包含多个 mini-batch，epoch 和 batch_size 的取值同样也可以修改，最终使用测试集来验证模型。代码如下所示。

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(train_features, train_labels, epochs=25,
batch_size=32, validation_data=(test_features, test_labels))
```

1.2.2 Argument Selection（参数选择）

为了对比神经网络在不同层数、不同神经元个数、不同激活函数条件下的性能表现，我引入了 8 组不同的网络架构参数，如表 1 所示。

表 1：神经网络架构参数选择

组	网络层数	神经元个数	激活函数	epoch	batch_size
1	1（隐藏层）	32	softmax	25	32
	2（隐藏层）	16	softmax		
	3（输出层）	4	softmax		
2	1（隐藏层）	32	relu	8	
	2（隐藏层）	16	relu		
	3（输出层）	4	softmax		
3	1（隐藏层）	64	relu	10	
	2（隐藏层）	32	relu		
	3（输出层）	4	softmax		
4	1（隐藏层）	32	tanh	10	
	2（隐藏层）	16	tanh		
	3（输出层）	4	softmax		
5	1（隐藏层）	32	sigmoid	25	
	2（隐藏层）	16	sigmoid		
	3（隐藏层）	8	relu		
	4（输出层）	4	softmax		
6	1（隐藏层）	32	sigmoid	15	
	2（隐藏层）	16	tanh		
	3（隐藏层）	8	relu		
	4（输出层）	4	softmax		
7	1（隐藏层）	8	sigmoid	40	
	2（隐藏层）	8	sigmoid		
	3（隐藏层）	8	relu		
	4（输出层）	4	softmax		
8	1（隐藏层）	128	sigmoid	25	
	2（隐藏层）	32	sigmoid		
	3（隐藏层）	16	sigmoid		
	4（隐藏层）	8	relu		
	5（输出层）	4	softmax		

1.3 Experimental Results（实验结果）

为了保存每一轮 mini-batch 训练后的模型在训练集和测试集上的损失(loss)，使用下面的语句运行 main.py，并将训练过程中程序输出到终端的结果保存在 output.log 文件中。

```
python main.py > output.log
```

据此，可以直接看出每次 mini-batch 训练后模型在训练集和测试集上的损失（Train loss 和 Test loss），以及其在训练集上的准确率 accuracy，如图 2 所示。由于日志文件很长，因此后续会将 loss 的变化趋势绘制为图片的形式进行展示。

1.3.1 第一组神经网络架构实验结果

在 epoch=25、batch_size=32 的条件下，训练完成后模型在训练集和测试集上的损失 loss 和准确率 acc 分别如图 3 所示。每个 mini-batch 中包含的训练样本数量为 batch_size。此时测试集的 accuracy 达到了 0.9275。

整个训练过程中，模型在训练集和测试集上的损失 loss 随 mini-batch 数量的变化如图 4 所示。

```
Epoch 1/25
    Mini-Batch: 0 - Train Loss: 1.392688512802124  - Test Loss: 1.3874133825302124

 1/113 [..............................] - ETA: 1:31 - loss: 1.3927 - accuracy: 0.3125     Mini-Batch: 1 - Train Loss: 1.3876265287399292  - Test Loss: 1.3874
 2/113 [..............................] - ETA: 8s - loss: 1.3876 - accuracy: 0.2338       Mini-Batch: 2 - Train Loss: 1.3853647708892822  - Test Loss: 1.3874
 3/113 [..............................] - ETA: 8s - loss: 1.3854 - accuracy: 0.2454      Mini-Batch: 3 - Train Loss: 1.3847947120666504  - Test Loss: 1.387549
 4/113 [>.............................] - ETA: 8s - loss: 1.3848 - accuracy: 0.2431      Mini-Batch: 4 - Train Loss: 1.389379858970642  - Test Loss: 1.3875761
 5/113 [>.............................] - ETA: 8s - loss: 1.3894 - accuracy: 0.2269      Mini-Batch: 5 - Train Loss: 1.3902151584625244  - Test Loss: 1.387582
 6/113 [>.............................] - ETA: 8s - loss: 1.3902 - accuracy: 0.2315      Mini-Batch: 6 - Train Loss: 1.3879114389419556  - Test Loss: 1.387572
 7/113 [>.............................] - ETA: 8s - loss: 1.3879 - accuracy: 0.2292      Mini-Batch: 7 - Train Loss: 1.3873651027679443  - Test Loss: 1.387571
 8/113 [=>............................] - ETA: 8s - loss: 1.3874 - accuracy: 0.2361      Mini-Batch: 8 - Train Loss: 1.3877942562103271  - Test Loss: 1.387537
 9/113 [=>............................] - ETA: 7s - loss: 1.3878 - accuracy: 0.2245      Mini-Batch: 9 - Train Loss: 1.386579990386963  - Test Loss: 1.3875069
10/113 [=>............................] - ETA: 7s - loss: 1.3866 - accuracy: 0.2338      Mini-Batch: 10 - Train Loss: 1.3866013288497925  - Test Loss: 1.38748
11/113 [=>............................] - ETA: 7s - loss: 1.3866 - accuracy: 0.2361      Mini-Batch: 11 - Train Loss: 1.3860275745391846  - Test Loss: 1.38748
12/113 [==>...........................] - ETA: 7s - loss: 1.3860 - accuracy: 0.2407      Mini-Batch: 12 - Train Loss: 1.3882582187652588  - Test Loss: 1.38747
13/113 [==>...........................] - ETA: 7s - loss: 1.3883 - accuracy: 0.2315      Mini-Batch: 13 - Train Loss: 1.3884354829788208  - Test Loss: 1.38744
```

图 2：output.log 记录了每个 mini-batch 训练后模型在训练集和测试集上的损失 loss

```
训练集损失train_loss： 0.3241255283355713
训练集准确率train_acc： 0.910277783870697
测试集损失test_loss： 0.29620134830474854
测试集准确率test_acc： 0.9275000095367432
```

图 3：实验结果（第一组）

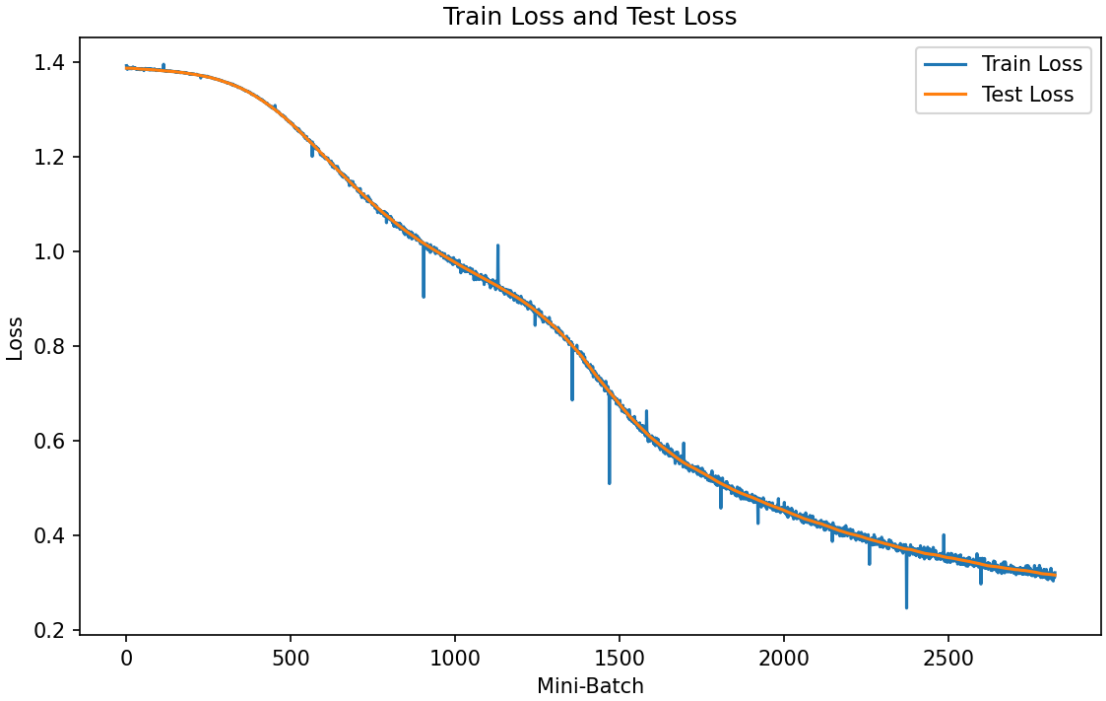

图 4：训练集和测试集上的损失 loss 随 mini-batch 数量的变化（第一组）

1.3.2 第二组神经网络架构实验结果

在 epoch=25、batch_size=32 的条件下，训练完成后模型在训练集和测试集上的损失 loss 和准确率 acc 分别如图 5 所示。此时测试集的 accuracy 达到了 0.9399。

整个训练过程中，模型在训练集和测试集上的损失 loss 随 mini-batch 数量的变化如图 6 所示。

```
训练集损失train_loss： 0.20550349354743958
训练集准确率train_acc： 0.9263888597488403
测试集损失test_loss： 0.2123061716556549
测试集准确率test_acc： 0.9399999976158142
```

图 5：实验结果（第二组，epoch=25）

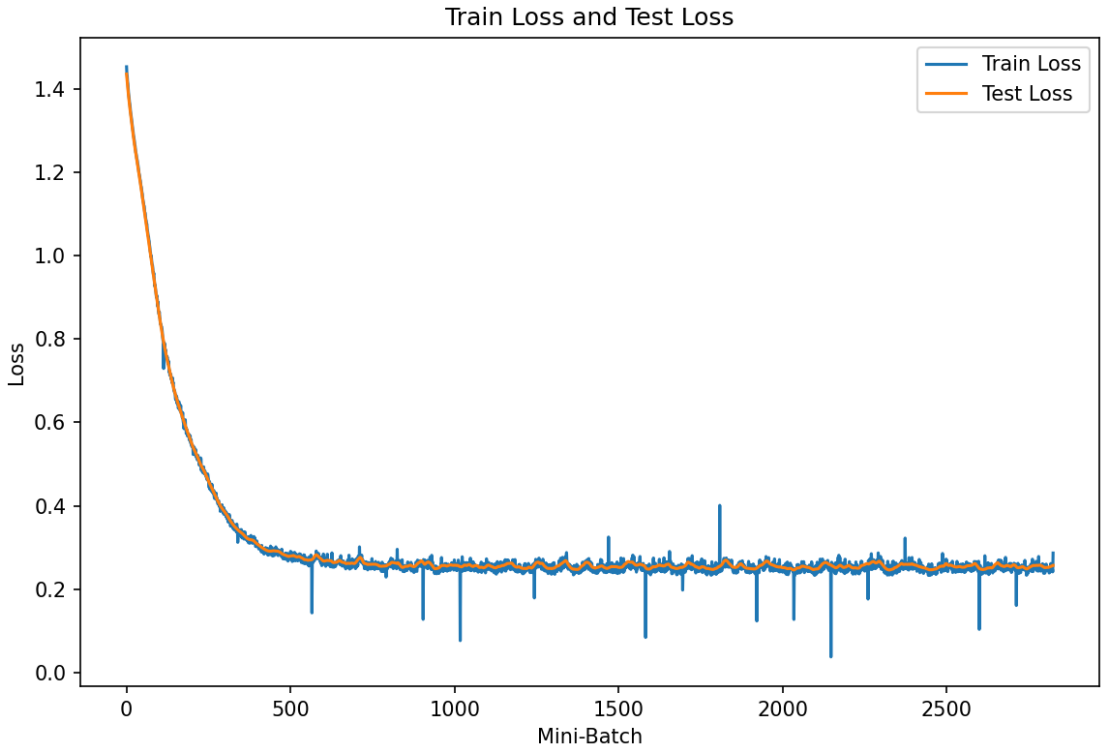

图 6：训练集和测试集上的损失 loss 随 mini-batch 数量的变化（第二组，epoch=25）

观察图 6 可以发现，在 mini-batch 达到第 500 批左右时，模型已经收敛。因此考虑将训练超参数中的 epoch 由 25 减少为 8，再次训练，预期可以得到相近的效果。同样地，训练完成后模型在训练集和测试集上的损失 loss 和准确率 acc 分别如图 7 所示，模型在训练集和测试集上的损失 loss 随 mini-batch 数量的变化如图 8 所示。此时测试集的 accuracy 达到了 0.9325。

```
训练集损失train_loss： 0.2206786572933197
训练集准确率train_acc： 0.9247221946716309
测试集损失test_loss： 0.20221111178398132
测试集准确率test_acc： 0.9325000047683716
```

图 7：实验结果（第二组，epoch=8）

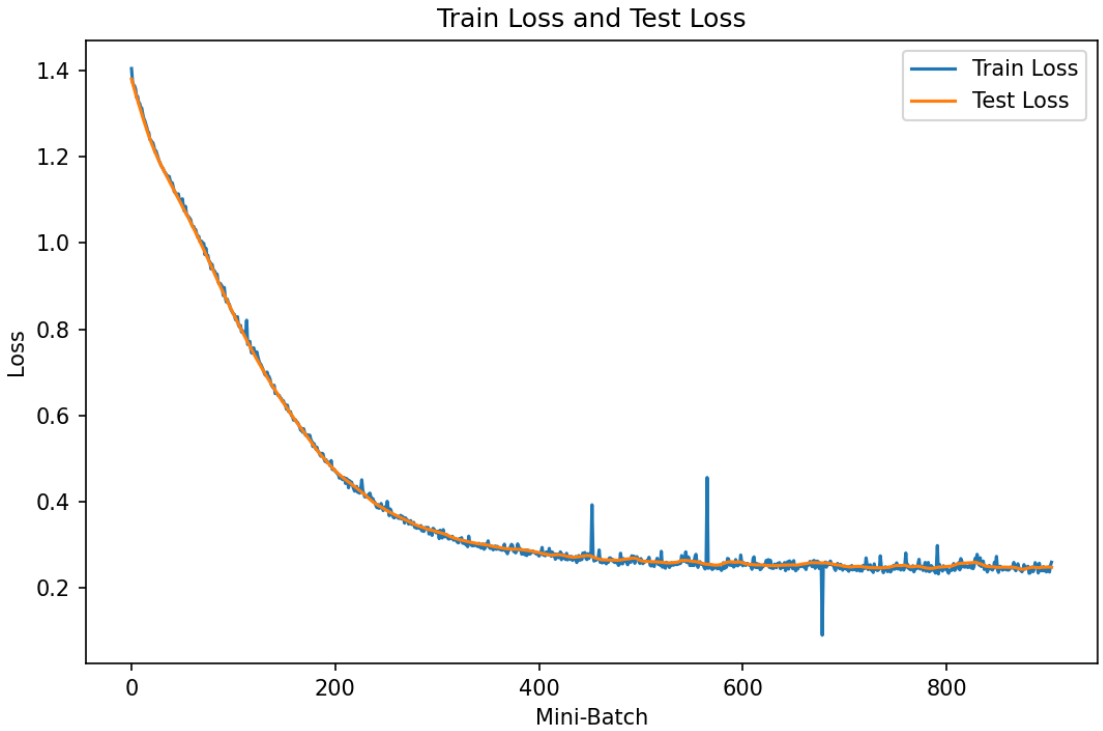

图 8：训练集和测试集上的损失 loss 随 mini-batch 数量的变化（第二组，epoch=8）

1.3.3 第三组神经网络架构实验结果

在 epoch=10、batch_size=32 的条件下，训练完成后模型在训练集和测试集上的损失 loss 和准确率 acc 分别如图 9 所示。此时测试集的 accuracy 达到了 0.9399。

整个训练过程中，模型在训练集和测试集上的损失 loss 随 mini-batch 数量的变化如图 10 所示。

```
训练集损失train_loss：  0.21129794418811798
训练集准确率train_acc：  0.926111102104187
测试集损失test_loss：  0.17863810062408447
测试集准确率test_acc：  0.9399999976158142
```

图 9：实验结果（第三组）

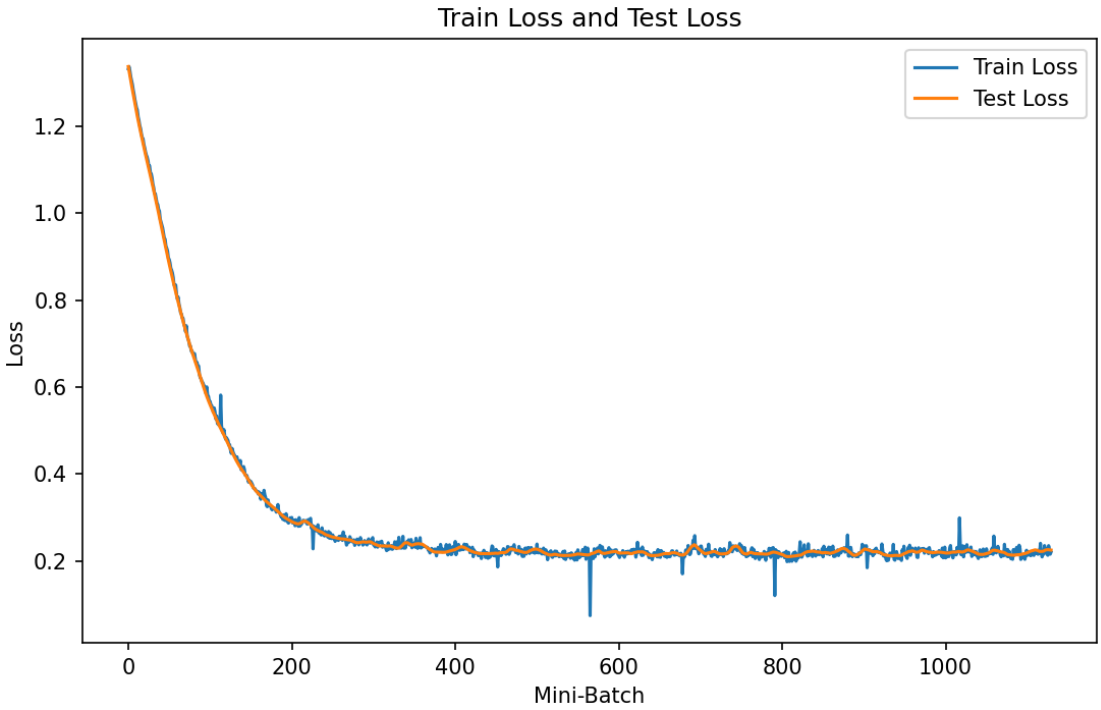

图 10：训练集和测试集上的损失 loss 随 mini-batch 数量的变化（第三组）

1.3.4 第四组神经网络架构实验结果

在 epoch=10、batch_size=32 的条件下，训练完成后模型在训练集和测试集上的损失 loss 和准确率 acc 分别如图 11 所示。此时测试集的 accuracy 达到了 0.9225。

整个训练过程中，模型在训练集和测试集上的损失 loss 随 mini-batch 数量的变化如图 12 所示。

```
训练集损失train_loss：  0.21937380731105804
训练集准确率train_acc：  0.9263888597488403
测试集损失test_loss：  0.2591528296470642
测试集准确率test_acc：  0.9225000143051147
```

图 11：实验结果（第四组）

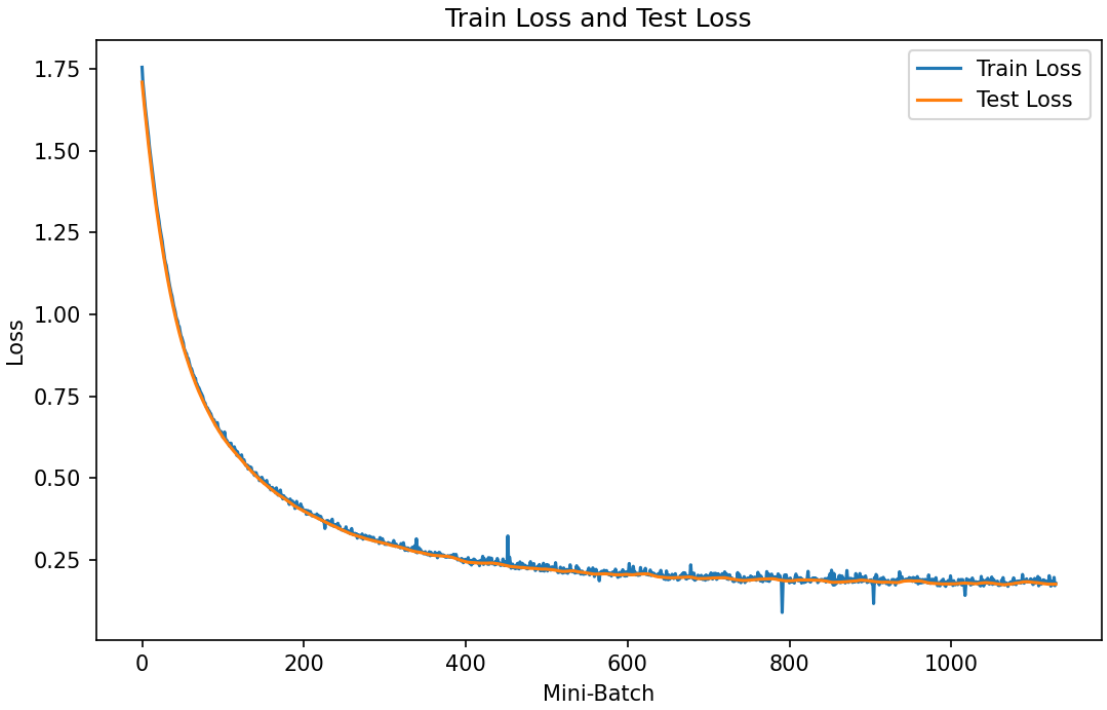

图 12：训练集和测试集上的损失 loss 随 mini-batch 数量的变化（第四组）

1.3.5 第五组神经网络架构实验结果

在 epoch=25、batch_size=32 的条件下，训练完成后模型在训练集和测试集上的损失 loss 和准确率 acc 分别如图 13 所示。此时测试集的 accuracy 达到了 0.9250。

整个训练过程中，模型在训练集和测试集上的损失 loss 随 mini-batch 数量的变化如图 14 所示。

```
训练集损失train_loss： 0.2197691798210144
训练集准确率train_acc： 0.9252777695655823
测试集损失test_loss： 0.23071099817752838
测试集准确率test_acc： 0.925000011920929
```

图 13：实验结果（第五组）

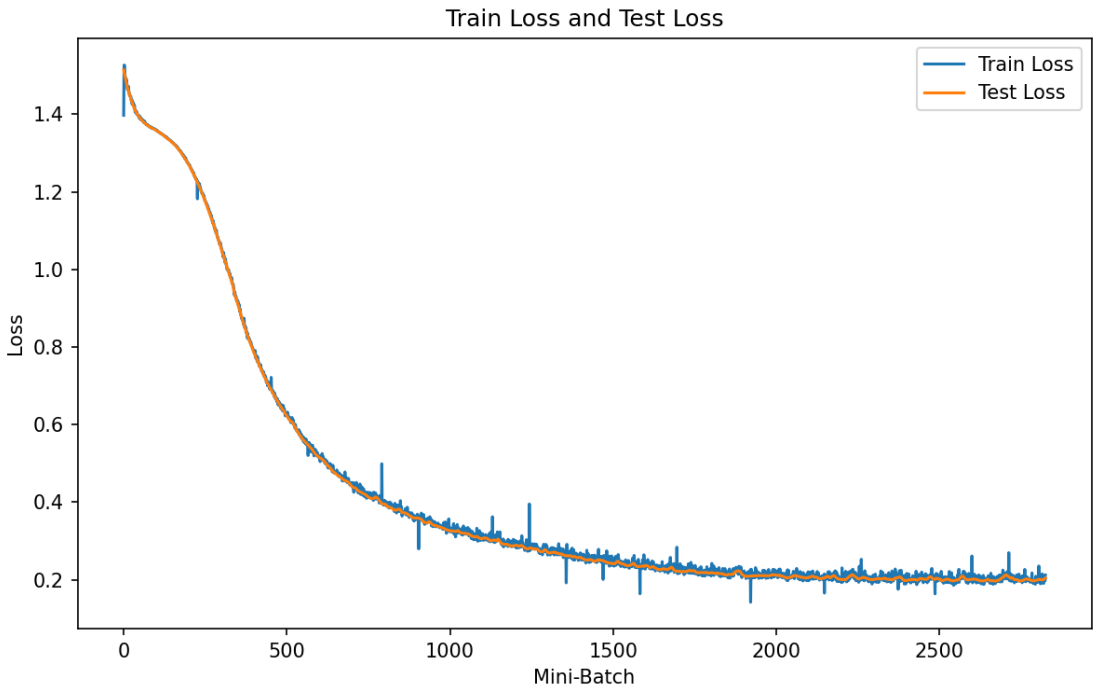

图 14：训练集和测试集上的损失 loss 随 mini-batch 数量的变化（第五组）

1.3.6 第六组神经网络架构实验结果

在 epoch=15、batch_size=32 的条件下，训练完成后模型在训练集和测试集上的损失 loss 和准确率 acc 分别如图 15 所示。此时测试集的 accuracy 达到了 0.9250。

整个训练过程中，模型在训练集和测试集上的损失 loss 随 mini-batch 数量的变化如图 16 所示。

```
训练集损失train_loss： 0.24214725196361542
训练集准确率train_acc： 0.9211111068725586
测试集损失test_loss： 0.24507245421409607
测试集准确率test_acc： 0.925000011920929
```

图 15：实验结果（第六组）

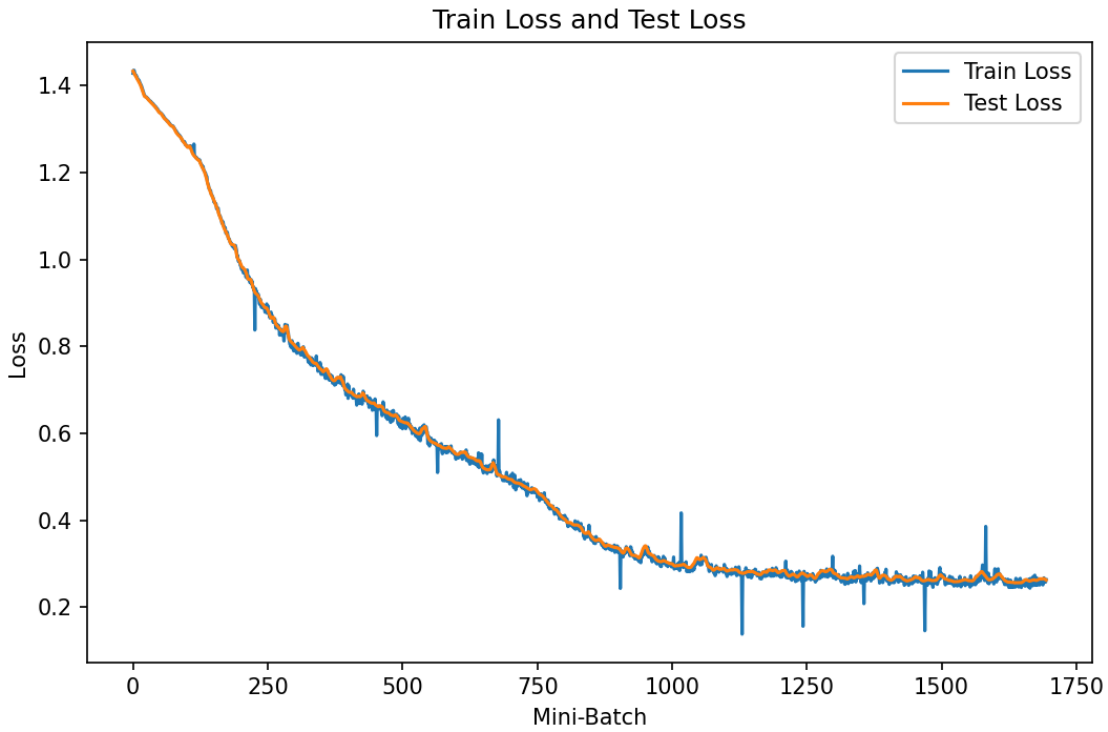

图 16：训练集和测试集上的损失 loss 随 mini-batch 数量的变化（第六组）

1.3.7 第七组神经网络架构实验结果

在 epoch=40、batch_size=32 的条件下，训练完成后模型在训练集和测试集上的损失 loss 和准确率 acc 分别如图 17 所示。此时测试集的 accuracy 达到了 0.8725。

整个训练过程中，模型在训练集和测试集上的损失 loss 随 mini-batch 数量的变化如图 18 所示。

```
训练集损失train_loss： 0.33074209094047546
训练集准确率train_acc： 0.8902778029441833
测试集损失test_loss： 0.3895834684371948
测试集准确率test_acc： 0.8725000023841858
```

图 17：实验结果（第七组）

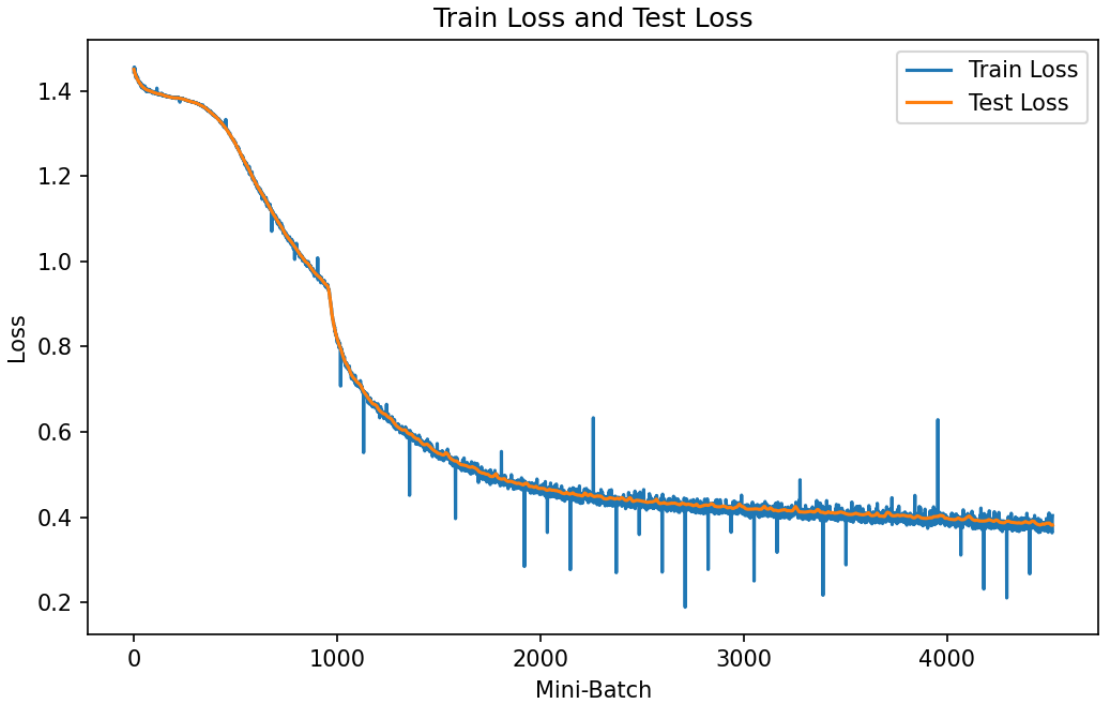

图 18：训练集和测试集上的损失 loss 随 mini-batch 数量的变化（第七组）

1.3.8 第八组神经网络架构实验结果

在 epoch=25、batch_size=32 的条件下，训练完成后模型在训练集和测试集上的损失 loss 和准确率 acc 分别如图 19 所示。此时测试集的 accuracy 达到了 0.9424。

整个训练过程中，模型在训练集和测试集上的损失 loss 随 mini-batch 数量的变化如图 20 所示。

```
训练集损失train_loss：  0.21322700381278992
训练集准确率train_acc：  0.9247221946716309
测试集损失test_loss：  0.2025243639945984
测试集准确率test_acc：  0.9424999952316284
```

图 19：实验结果（第八组）

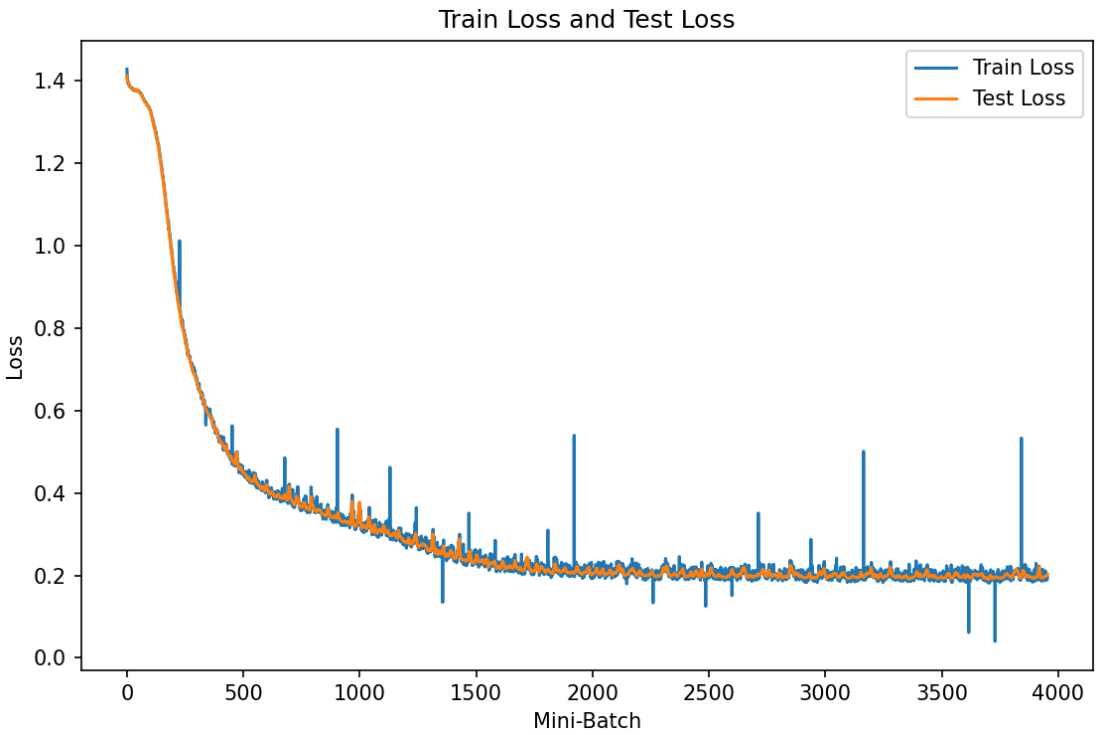

图 20：训练集和测试集上的损失 loss 随 mini-batch 数量的变化（第八组）

1.4 Discussion（结果分析）

1.4.1 总体结果分析

在上述 8 组神经网络架构下，通过选择合理的训练参数（epoch 和 batch_size），绝大部分情况下在测试集上的 accuracy 都可以达到 0.92 以上，这说明在训练模型时，有很多种不同的神经网络结构都可以满足要求。在实际训练过程中，需要根据训练效果及时调整网络层数、各层的神经元个数、激活函数以及 epoch 数量等。

观察上面的训练集和测试集上的损失 loss 随 mini-batch 数量的变化趋势图像，可以发现其共同特征是模型在训练集和测试集上的损失 loss 都是相近的（橙色线和蓝色线基本重合），但模型在测试集上的 loss 波动较小，在训练集上的 loss 可能会有较大的突变现象，但总体趋势还是持续下降。这两者不会无限度地下降，在训练轮数达到一定值后就会趋于稳定，这标记着模型已经接近收敛。

1.4.2 不同组网络架构之间的对比

（1）第 1、2 组网络架构结果对比

第一组架构下，前两个隐藏层使用的激活函数均为 softmax，而在第二组架构下，前两个隐藏层使用的激活函数均为 relu。

通过观察并对比图 4 和图 8，可以发现**激活函数不同，模型收敛的速度也不同**。使用 softmax 激活函数时，模型训练了 2500 个 mini-batch 后仍然没有观察到明显的收敛；而使用 relu 激活函数时，模型训练了 600 个 mini-batch 后基本上就收敛了。

（2）第 2、3 组网络架构结果对比

第二组架构下，前两个隐藏层的神经元个数分别为 32、16，而在第三组架构下，前两个隐藏层的神经元个数分别为 64、32。

通过观察并对比图 8 和图 10，可以发现**神经元个数不同，模型收敛的速度也不同，神经元个数较多时，收敛速度较快**。在这里，神经元个数为 32、16 时，模型在 600 个 mini-batch 后收敛；而神经元个数为 64、32 时，模型在 300 个 mini-batch 后就收敛了。这说明在一定范围内，更多的神经元可以更好地处理输入特征，在更少的训练迭代后达到比较好的效果。

（3）第 2、4 组网络架构结果对比

在第四组架构下，前两个隐藏层使用的激活函数均为 tanh。对比可知，这两组最后在测试集上的 accuracy 基本相同，并且模型达到收敛所用的 batch 数量也基本相同。因此 **tanh 和 relu 激活函数在本任务中都是有效的**。

（4）第 5、7 组网络架构结果对比

第七组架构相对于第五组，主要的区别是前两个隐藏层的神经元个数都变少了。通过对比图 13 和图 17，可以发现第七组的模型在测试集的准确率相比第五组有所下降。因此可以推测，**神经元数量的减少会导致对特征的学习不充分，并使预测准确率下降**。

（5）第 5、8 组网络架构结果对比

第八组架构相对于第五组，主要的区别是在最前面添加了一层含有 128 个神经元、使用 sigmoid 作为激活函数的隐藏层。

通过对比图 14 和图 20，可以发现神经网络隐藏层层数不同，模型收敛的速度也不同。第八组的模型在第 2300 个 batch 左右收敛，而第五组的模型在第 2500 个 batch 左右收敛。

1.4.3 各类激活函数的性质

综合以上分析，可以对各类激活函数的性质进行简要的描述。

（1）隐藏层常见的激活函数

①relu 激活函数

函数表达式为：$\mathrm{Re}\,LU(x)=\begin{cases}x, x\geq 0\\ 0, x<0\end{cases}=\min(x,0)$

它可以在一定程度上缓解梯度消失的问题，计算上更加高效，收敛更快。

②sigmoid 激活函数

函数表达式是：$\sigma(x)=\dfrac{1}{1+e^{-x}}$

它可以将输入映射到(0, 1)之间的概率分布，是一种饱和函数，收敛相对较慢。

③tanh 激活函数

函数表达式是：$\tanh(x)=\dfrac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

它可以将输出映射到(-1, 1)之间的概率分布，是一个零中心化的函数。

（2）输出层常见的激活函数

①二分类问题

常用的是 sigmoid 激活函数。在二分类问题中，sigmoid 的本质就是去拟合伯努利分布（0-1 分布），也就是表达正面事件和反面事件发生的概率。输出层只需要根据 sigmoid 函数的输出即可找出可能性最大的情形。

同样地，tanh 激活函数也可以应用于二分类问题。

②多分类问题

常用的激活函数包括 softmax 激活函数和 sigmoid 激活函数。

对于 softmax 函数，其函数表达式是：$Softmax(x_i)=\frac{e^{x_i}}{\sum_j e^{x_j}}$，其中 x 是一个标签向量，在本实验中就是$[0,1,2,3]$，x^i 和 x^j 是向量中的两个元素。该函数可以预测每个类别的概率，所有的概率之和为 1，最终输出层将概率最高的类别输出。

对于 sigmoid 函数，它基于二分类问题，在多分类问题中，它需要使用多次二元分类，即 sigmoid 输出的数值表示是否属于该类。

1.5 Reference（参考文献）

[1] 华中科技大学计算机学院 2023 年秋季计算机视觉《第四讲 前馈神经网络》

1.6 Appendix（源代码）

1.6.1 main.py

作用：定义神经网络，训练模型。
代码：

```
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.callbacks import Callback

class LossHistory(Callback):
    def __init__(self, train_data, test_data):
        self.train_data = train_data
        self.test_data = test_data

    def on_batch_end(self, batch, logs=None):
        train_loss = logs.get('loss')
        test_loss = self.model.evaluate(self.test_data[0],
self.test_data[1], verbose=0)
        print("     Mini-Batch:", batch, "- Train Loss:", train_loss, "
- Test Loss:", test_loss[0])

    def on_epoch_end(self, epoch, logs=None):
        pass

data = pd.read_csv('dataset.csv')

# 随机打乱数据集
data = data.sample(frac=1).reset_index(drop=True)

train_size = int(0.9 * len(data))
train_data = data[:train_size]
test_data = data[train_size:]

train_features = train_data[['data1', 'data2']].to_numpy()
```

```
train_labels = train_data['label'].to_numpy() - 1
test_features = test_data[['data1', 'data2']].to_numpy()
test_labels = test_data['label'].to_numpy() - 1

# 定义神经网络架构，网络层数、激活函数、神经元个数可修改
model = tf.keras.Sequential([
    tf.keras.layers.Dense(16, activation='sigmoid', input_shape=(2,)),
    tf.keras.layers.Dense(16, activation='sigmoid'),
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dense(4, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(train_features, train_labels, epochs=25,
batch_size=32, validation_data=(test_features, test_labels),
callbacks=[LossHistory((train_features, train_labels), (test_features,
test_labels))])

train_loss, train_accuracy = model.evaluate(train_features,
train_labels, verbose=0)
test_loss, test_accuracy = model.evaluate(test_features, test_labels,
verbose=0)

print("训练集损失 train_loss：", train_loss)
print("训练集准确率 train_acc：", train_accuracy)
print("测试集损失 test_loss：", test_loss)
print("测试集准确率 test_acc：", test_accuracy)
```

1.6.2 extract.py

作用：使用正则表达式提取每个 batch 训练后再训练集和测试集上的损失。

代码：

```
import re
import csv
import chardet

input_file = "output.log"
output_file = "Loss.csv"
loss_values = []
```

```
with open(input_file, 'rb') as file:
    result = chardet.detect(file.read())
encoding = result['encoding']
with open(input_file, 'r', encoding=encoding) as file:
    for line in file:
        match = re.search(r'Train Loss: (\d+\.\d+)  - Test Loss: 
(\d+\.\d+)', line)
        if match:
            train_loss = float(match.group(1))
            test_loss = float(match.group(2))
            loss_values.append([train_loss, test_loss])

with open(output_file, 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['Train Loss', 'Test Loss'])
    for values in loss_values:
        writer.writerow(values)
```

1.6.3 draw.py

作用：绘制 Train Loss 和 Test Loss 的变化趋势图。
代码：

```
import csv
import matplotlib.pyplot as plt

input_file = 'Loss.csv'
train_loss = []
test_loss = []
with open(input_file, 'r') as file:
    reader = csv.reader(file)
    next(reader)
    for row in reader:
        train_loss.append(float(row[0]))
        test_loss.append(float(row[1]))

plt.plot(train_loss, label='Train Loss')
plt.plot(test_loss, label='Test Loss')
plt.xlabel('Mini-Batch')
plt.ylabel('Loss')
plt.title('Train Loss and Test Loss')
plt.legend()

plt.show()
```